VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

INOVACE LABORATORNÍ ÚLOHY - "TANKY"
LABORATORY MODEL „LIQUID TANKS“ INNOVATION

BAKALÁŘSKÁ PRÁCE
SEMESTRAL THESIS

AUTOR PRÁCE
AUTHOR
Antonín Škapa

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Václav Kaczmarczyk, Ph.D.

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: Antonín Škapa ***ID:*** 172136
Ročník: 3 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Inovace laboratorní úlohy - "Tanky"

POKYNY PRO VYPRACOVÁNÍ:

Proveďte komplexní inovaci laboratorní úlohy "Tanky", která je hlavní součástí cvičení předmětu Automatizace Procesů.

1. Seznamte se s novou koncepcí úlohy
2. Zadokumentujte navržený zdrojový kód úlohy
3. Na základě získaných poznatků proveďte úpravy stávající dokumentace k předmětu.
4. Vytvořte zkrácenou verzi dokumentace obsahující pouze přehledně sestavené výpisy kódů, jako podklad pro vyučující, kteří budou hotové práce hodnotit.

DOPORUČENÁ LITERATURA:

Pásek, J.: Automatizace procesů - Laboratorní cvičení I

Termín zadání: 5. 2. 2018 ***Termín odevzdání:*** 21.5.2018

Vedoucí práce: Ing. Václav Kaczmarczyk, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta elektrotechniky a komunikačních technologií
Ústav automatizace a měřicí techniky

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Abstrakt

Tato práce se zabývá inovací laboratorní úlohy „Tanky“. Dále jsou zde rozebrány programovací jazyky pro PLC, zejména SCL.

Klíčová slova

PLC, Siemens S7, SCL, ST, Laboratorní úloha, HMI

Abstract

This bachelor thesis deals with innovation of laboratory exercise „Tanks“. Furthermore there are briefly analyzed PLC programming languages, especially SCL.

Keywords

PLC, Siemens S7, SCL, ST, Laboratory exercise, HMI

Antonín Škapa. Inovace laboratorní úlohy „Tanky“.Brno:Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace, 2018. 30 s., 3 s. příloh. Bakalářská práce. Vedoucí práce: Ing.Václav Kaczmarczyk, Ph.D.

Prohlášení

Prohlašuji, že svoji bakalářskou práci na téma Inovace laboratorní úlohy „Tanky“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené semestrální práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

(podpis autora)

Poděkování

Děkuji vedoucímu bakalářské práce Ing.Václavu Kaczmarczykovi, Ph.D.za odborné vedení, konstruktivní kritiku a přátelský přístup. Dále bych také chtěl poděkovat Ing.Tomáši Benešlovi za neúnavné konzultace, praktické ukázky použitého softwaru a užitečné tipy zkušeného programátora.

Obsah

Seznam obrázků

1 ÚVOD

Cílem této bakalářské práce bylo provést inovaci laboratorní úlohy tanky do předmětu MAUP dle standartu BATCH v aktuální TIA portálu v14 SP1. Spolu s vedoucím práce jsme stanovili konkrétní cíle a vlastnosti tohoto projektu. Projekt byl průběžně testován v laboratoři na reálném PLC řady S7-1500 a HMI TP-700 Comfort.

- Seznamte se se novou koncepcí úlohy
- Zadokumentujte navržený zdrojový kód úlohy
- Na základě získaných poznatků proveďte úpravu stávající dokumentace k předmětu
- Vytvořte zkrácenou verzi dokumentace obsahující pouze přehledně sestavené výpisy kódů, jako podklad pro vyučující, kteří budou hotové práce hodnotit.

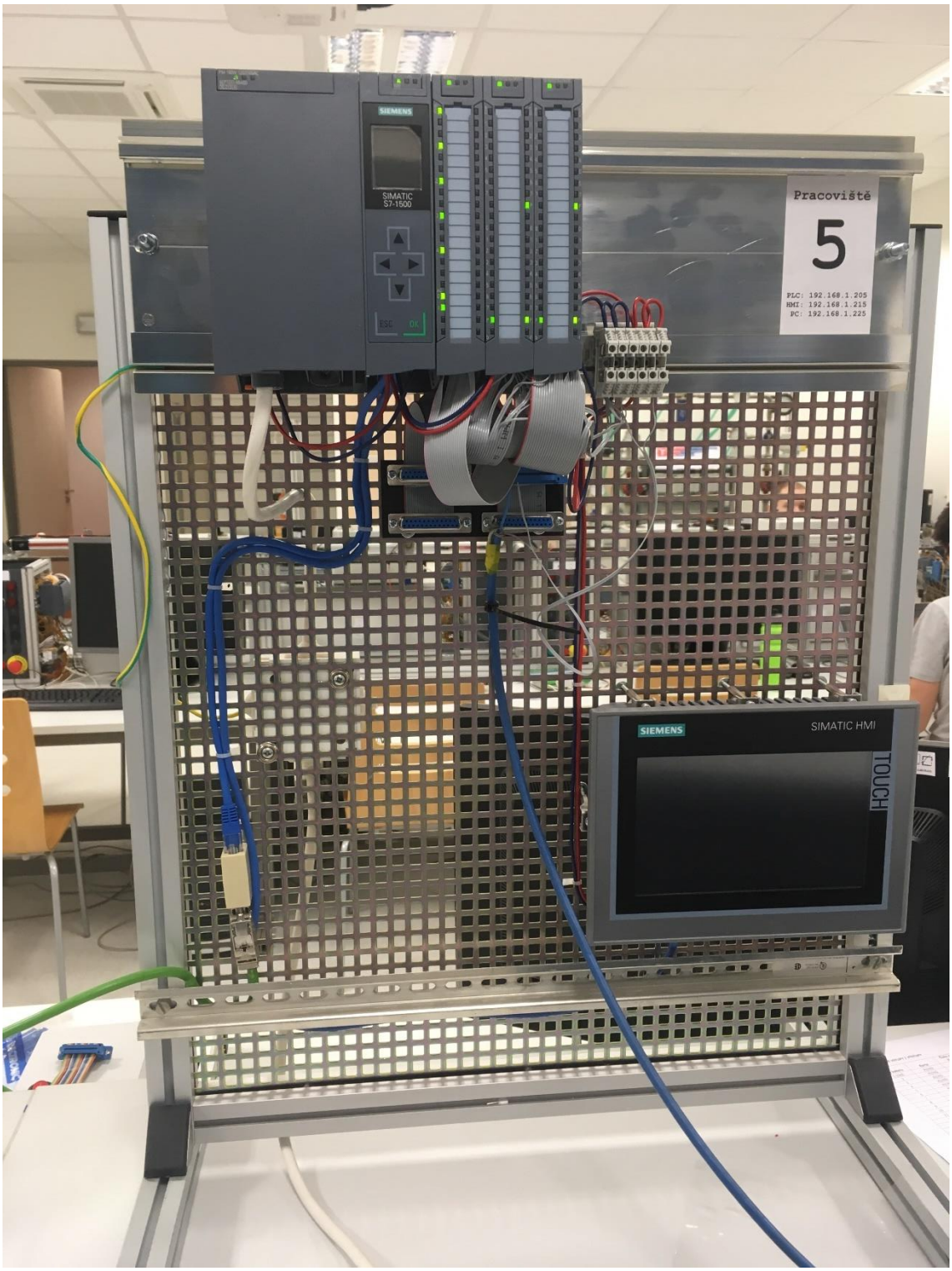
1.1 Obsah přílohy

V příloze se nachází nová dokumentace k předmětu MAUP dle zadání, výpis kódu pro potřeby hodnocení jednotlivých projektů a „tahák“ se základní strukturou SCL.

1.2 Stručný popis

Celá bakalářská práce se skládá ze čtyřech kapitol. Ve druhé kapitole se zabývám jednotlivými použitými programovacími jazyky a jejich výhodami a nevýhodami. Nejvíce je zde rozebrán jazyk SCL, který je použit pro celou tuto úlohu. Vybrané použití jazyka SCL demonstruji na modelových příkladech. V kapitole třetí popisuji použité PLC a přídavné karty a v kapitole čtvrté dochází k zhodnocení dosavadní práce. Samotná dokumentace, která byla hlavní částí práce je v příloze.

1.3 Náhled pracoviště



Obr. 1-Náhled pracoviště

2 PROGRAMOVACÍ JAZYKY

V TIA Portálu 14 můžeme programovat v jazycích Statement list(STL), Structure control language(SCL), Ladder diagram(LAD), Fucntion block diagram(FDB) a GRAPH. Každý z nich se hodí pro konkrétní typ úlohy. V semestrální práci jsem použil jazyky STL,GRAPH(Grafcet) a LAD. Naproti tomu v bakalářské práci již byl použit pouze jazyk SCL. [2]

2.1 STL

Statement list-můžeme přeložit jako seznam instrukcí. Jedná se o textový jazyk přizpůsobený pro použití v PLC. Používá mnemotechnické instrukce jako například A-AND, L-LOAD atd. Jedná se o nejnižší úroveň programování, ale přesto můžeme u mnoha aplikací napsat v tom jazyce kód nejrychleji a nejefektivněji. Jedná se o jazyk normy IEC 61131-3 a standartu IEC 60848. [1] [2]

2.2 GRAFCET

Grafcet je grafický programovací jazyk, který je nejvhodnější pro programování sekvenčních dějů, jelikož zde definujeme pouze úkony v aktuálním stavu a podmínky pro přechod do stavu jiného. Po vykonání posledního stavu se program opět vrací do stavu prvního a začíná znova. Princip fungování vychází z Petriho sítí. Jedná se o jazyk normy IEC 61131-3 a standartu IEC 60848. [3] [1] [11]

2.3 LAD

Ladder diagram-takzvané spínačové zobrazení vychází z principu elektrických schémat v liniovém zapojení, proto je program v tomto jazyku je asi nejlépe čitelný a pochopitelný pro uživatele, který se s PLC ještě nesetkal. Pro složité programy s mnoha větvemi ale není tolik vhodný, jelikož se program stává nepřehledným. Jedná se o jazyk normy IEC 61131-3 a standartu IEC 60848. [1][11]

2.4 SCL

Pokročilý strukturovaný jazyk na bázi Pascalu. Program píšeme ve formě strukturovaného textu s použitím metod vyšších programovacích jazyků jako například deklarací, podmínek, knihoven a dalších. Na druhou stranu používáme také prostředky jazyků LAD A STL jako například vstupy/výstupy, časovače, bitová paměť a další.

2.4.1 Vývoj jazyka SCL

Programovací jazyk Structured text byl vydán roku 1993 v normě IEC 61131-3, jejíž cílem bylo standardizovat programovací jazyky používané pro programování PLC. Structured text jako velmi intuitivní a výkonný programovací jazyk využívá typických operací jako je: logické větvení, smyčky a další. [8] Programy v SCL mohou být psány v jakémkoli textovém editoru a protože jejich jasná a přehledná syntaxe připomíná věty, tak je následné testování programu a celkové porozumění kódu velmi snadné. Díky tomu je tento jazyk vhodný pro komplexnější úlohy, kde by při použití jiných jazyků utrpěla celková přehlednost kódu a nebo by vůbec nebylo možné zadaný úkol efektivně realizovat.

Jak se postupně celý obor programování PLC přirozeně vyvíjí, tak je zde čím dál větší důraz na multiplatformnost použitých programovacích jazyků a hardwaru. Z tohoto důvodu nabývá jazyk SCL na čím dál větší oblibě, jelikož drtivá většina výrobců automatizační techniky tento jazyk podporuje a navíc není problém například provádět online změny v programu bez zastavení PLC. [10] Díky tomu jsou firmy schopny v případě potřeby měnit programovatelné automaty od jednoho výrobce za programovatelné automaty druhého výrobce znatelně snáz, popřípadě dolaďovat program za chodu linky. [8]

Z podstaty SCL plyne, že programátoři, kteří se již setkali s jazyky jako například C, Python atd. mají předpoklad se naučit SCL velice rychle. Navíc v TIA Portal 14 je mnoho již předpřipravených funkcí zajišťujících například komunikaci a nebo matematické funkce. Je zde také možné propojovat funkční bloky napsané v různých programovacích jazycích, což dává programátorovi velkou volnost a není tak nucen držet se jednoho jazyku.

2.4.2 Deklarace proměnných

SCL umožňuje deklaraci proměnných, podobně jako například v jazyce C a nebo symboly v jazyce LAD. Všechny jména proměnných musí začínat písmenem, ale další symboly již mohou obsahovat i čísla a některé speciální znaky. [8] Velice důležité je to, že Structured text není takzvaně „case-sensitive“, což znamená, že nerozeznává velké písmeno od malého. V rámci zachování programátorských konvencí, je ale dobré se držet zažitých pravidel pro pojmenovávání funkcí a proměnných. Proměnné můžeme deklarovat pomocí následujících klíčových slov:

- **VAR/END_VAR** Začátek/konec deklarace proměnných
- **VAR_GLOBAL** Globální proměnná

- **VAR_INPUT** Vstupní proměnná
- **VAR_OUTPUT** Výstupní proměnná
- **VAR_IN_OUT** Vstupně výstupní proměnná
- **VAR_ACCESS** Přímý přístup do paměti
- **VAR_EXTERNAL** Tímto klíčovým slovem se odkazujeme na globální proměnnou. Toto většinou není potřeba, jelikož to za programátora udělá překladač. Nejčastější použití je při odkazu například do knihovny nebo mimo aktuální rozsah překladu.
- **VAR_TEMP** Dočasná proměnná
- **AT** Pomocí tohoto klíčového slova můžeme definovat, kde přesně v paměti se bude proměnná nacházet
- **CONSTANT** Neměnná konstanta
- **RETAIN** Proměnná, která zůstane v PLC uložena i po jeho vypnutí a vypnutí napájení.
  [8]

2.4.3 FOR cyklus

Typická konstrukce FOR cyklu vypadá následovně:

„

```
FOR count := initial_value TO final_value BY increment DO
  <statement>;
END_FOR;
```

„[6]

Initial_value-výchozí hodnota od které počítáme FOR cyklus
Final_value-hodnota do které běží FOR cyklus
Increment-velikost kroku
<statement>-vykonávaný kód

Protože FOR cyklus má omezený, dopředu daný počet opakování, tak se používá tam, kde přesně takhle vlastnost může být výhodná, či přímo klíčová. Například v modelovém příkladu, kde máme za úkol namočit výrobek do kyseliny 10krát, tak přesně zde bude FOR cyklus výhodný.

2.4.4 WHILE cyklus

Oproti FOR cyklu, má WHILE cyklus formálně jednodušší zápis:

„

```
WHILE [boolean expression] DO

  <statement>;

END_WHILE;
```

„[7]

Boolean expression-True/false výraz při jehož splnění se bude cyklus provádět
<statement>-Samotný kód cyklu

WHILE cyklus se oproti FOR cyklu používá tam, kde je žádoucí, aby se výraz v cyklu opakovat až do splnění zadané podmínky. Pokud si zde opět pomůžeme modelovým příkladem z průmyslu, tak to bude tento: máme za úkol vkládat opakovaně odporový drát do lázně s kyselinou, dokud jeho odpor nedosáhne požadované úrovně. Z důvodu změny teploty, koncentrace kyseliny a dalších faktorů nebude potřebný počet cyklů vždy stejný, proto zde bude WHILE cyklus optimálně použit.

2.4.5 REPEAT cyklus

Posledním standartním typem cyklu je REPEAT cyklus:

„

```
REPEAT
  <statement>;
UNTIL [boolean expression]
END_REPEAT
```

„[5]

<statement>-Samotný kód cyklu
Boolean expression-True/false výraz při jehož splnění se cyklus **přeruší**

REPEAT cyklus funguje přesně opačným způsobem než WHILE cyklus. Všimněme si zde toho, že podmínka je zde až za prováděným kódem a proto se kód za všech okolností provede minimálně jednou. Modelovým příkladem by zde mohlo být načítání stavu zásobníku, dokud nenačteme nenulové, kladné, celé číslo. V tomto případě musíme provést načtení minimálně jednou a právě pro tento případ bude REPEAT cyklus ideálním kandidátem.

2.4.6 Použití EXIT v SCL

Pokud potřebujeme ukončit cyklus při určité podmínce i bez dokončení cyklu dle původně zadané podmínky, tak se nám výborně hodí příkaz EXIT, který zapouzdříme do IF podmínky.

„

```
WHILE current_level <= setpoint DO
Valve_Open := TRUE;
IF Sensor_Fault = TRUE THEN
Valve_Open := FALSE;
EXIT;
END_IF;
END_WHILE;
```

„[8]

Na modelovém příkladu výše je znázorněn kód pro WHILE cyklus, který má za úkol držet otevřený ventil, dokud není dosaženo požadované hladiny a nebo nezaznamenáme chybu sensoru. V případě chyby sensoru ventil okamžitě zavřeme a vystoupíme z WHILE cyklu.

2.4.7 Použití RETURN v SCL

RETURN se na rozdíl od EXIT používá uvnitř funkce a nebo funkčního bloku. [8] Používáme ho obvykle, pokud potřebujeme z funkce vystoupit před jejím koncem a obvykle je, jako EXIT, zapouzdřený v IF podmínce.

„

```
FUNCTION_BLOCK Valve_Control
VAR_INPUT
Tank_Level, Max_Level : REAL;
END_VAR
VAR_OUTPUT
Valve_Open : BOOL;
Out_Msg : STRING;
END VAR
IF Tank_Level > Max_Level THEN
Valve_Open := FALSE;
Out_Msg := 'Tank Level is Full';
RETURN;
 ELSE
Valve_Open := TRUE;
END_IF;
IF Tank_Level > Max_Level * 0.8 THEN
Out_Msg := 'Tank Level is High';
ELSE
```

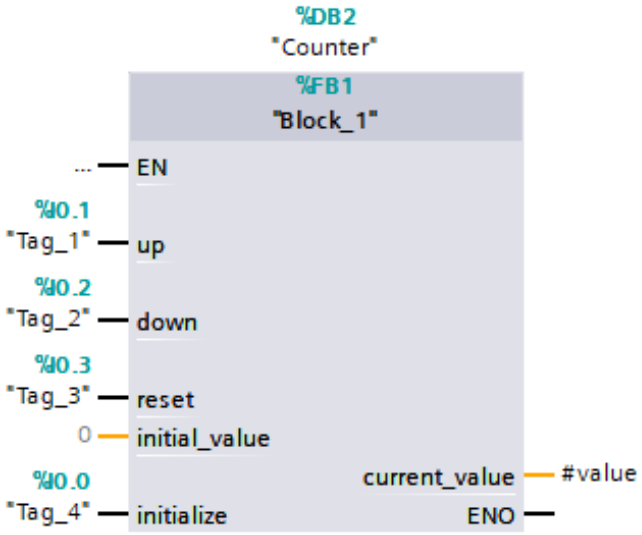
```
Out_Msg := 'Tank Level is Normal';
END_IF;
END_FUNCTION_BLOCK
```

„[8]

Na modelovém programu výše je napsán funkční blok, který otevírá a zavírá ventil podle stavu hladiny v nádrži. Pokud je nádrž naplněna, tak se ventil uzavře, je vyhozena hláška o naplnění nádrže a blok je ukončen. V případě, že se tak nestane, tak je stav hladiny dále zhodnocen a je vyhozena odpovídající hláška.

2.4.8 Čítač v jazyku SCL

V příkladu níže je ukázka použití čítače v jazyku SCL. Jedná se o modelový příklad, kdy pomocí čítače počítáme počet vyrobených výrobků na dvou výrobních linkách. PLC je připojeno na výrobní linky, které dávají při projetí vyrobeného výrobku linkou číslo 1. signál logická 1 na vstup I.0.1, naopak při zaznamenání výrobku linkou číslo 2. vidíme logickou 1 na vstupu I0.2. Můžeme tak v proměnné *value* sledovat zda-li obě linky mají stejnou výkonnost. První linka totiž k číslu v čítači jedničku přičítá a druhá odečítá. Vstup I0.3 slouží k vynulování čítače po proběhlé várce a logická 1 na vstupu I0.4 nám čítač znovu připraví nahráním startovací hodnoty. Na obrázku 2. můžeme vidět funkci samotného čítače přidanou do bloku OB1, který se spouští automaticky při zapnutí PLC.



Obr. 2-Hotový blok Counter

Na obrázku 3. jsou proměnné samotné funkce čítač napsané v programovacím jazyku SCL. Jelikož v našem příkladu potřebujeme hodnotu přičítat i odečítat je

použit čítač CTUD(Counter-Up/Down). Pokud se to hodí naší aplikaci, můžeme také použít čistě přičítací, nebo odečítací čítače, jejichž syntaxe je podobná.

Block_1

		Name	Data type	Default value	Retain
1		Input			
2		up	Bool	false	Non-retain
3		down	Bool	false	Non-retain
4		reset	Bool	false	Non-retain
5		initial_value	Int	0	Non-retain
6		initialize	Bool	false	Non-retain
7		<Add new>			
8		Output			
9		current_value	Int	0	Non-retain

Obr. 3-Proměnné bloku Counter

```
"Counter1".CTUD(CU:=#up,
                CD:=#down,
                R:=#reset,
                LD:=#initialize,
                PV:=#initial_value,
                CV=>#current_value     );
```

2.4.9 Časovač v jazyku SCL

Modelový příklad časovače ukazuje, jakým způsobem můžeme pomocí jednoho časovače řídit více událostí. V našem případě se jedná o postupné zapínání třech LED diod, každé po pěti sekundách. Výstup LED diod je ve formě struktury, na kterou už můžeme přímo navázat odpovídající fyzické výstupy. V příkladu vidíme použitý časovač TONR, jehož vstupy jsou IN-povolovací bit časovače, RESET-resetování časovače, PT-čas, po jehož uplynutí se výstup Q překlopí do stavu logická jedna a nakonec ET-aktuální uběhlý čas od spuštění časovače.

Pod kódem pro samotný časovač se nachází povel pro restartování časovače po uplynutí 4*5 sekund. Toto nám ve spolupráci s následující konstrukcí CASE zajistí, že LED diody se budou postupně rozsvěcovat s časovým intervalem 5 sekund a následně zhasnou a celý proces se bude opakovat.

Následně přepočítáváme čas na hodnotu int, kterou dělíme pěti sekundami (čas mezi sepnutím dvou LED diod). Funkce FLOOR nám zajistí zaokrouhlení hodnoty na celé číslo směrem dolů a jelikož tato funkce má na vstupu typ real, musíme podělenou hodnotu nejdříve na tento typ převést.

Ihned poté následuje konstrukce CASE zajišťující samotné zapínání a vypínání podle čísla přítomného v proměnné led.

```
"IEC_Timer_0_DB".TONR(IN:=#enable,
                      R:=#reset,
                      PT:=#time2go,
                      Q=>#out,
                      ET=>#time);
IF #time > (#time2go * 4) THEN
    #reset := 1;
    ;
END_IF;

#led := FLOOR(INT_TO_REAL(TIME_TO_INT(#time) /
TIME_TO_INT(T#5S)));
CASE #led OF
    1:
        #output_led."1" := 1;
        #output_led."2" := 0;
        #output_led."3" := 0;
        ;
    2:
        #output_led."1" := 1;
        #output_led."2" := 1;
        #output_led."3" := 0;
        ;
    3:
        #output_led."1" := 1;
        #output_led."2" := 1;
        #output_led."3" := 1;
        ;
        ;
    ELSE
        #output_led."1" := 0;
        #output_led."2" := 0;
        #output_led."3" := 0;
        ;
END_CASE;
```

timer

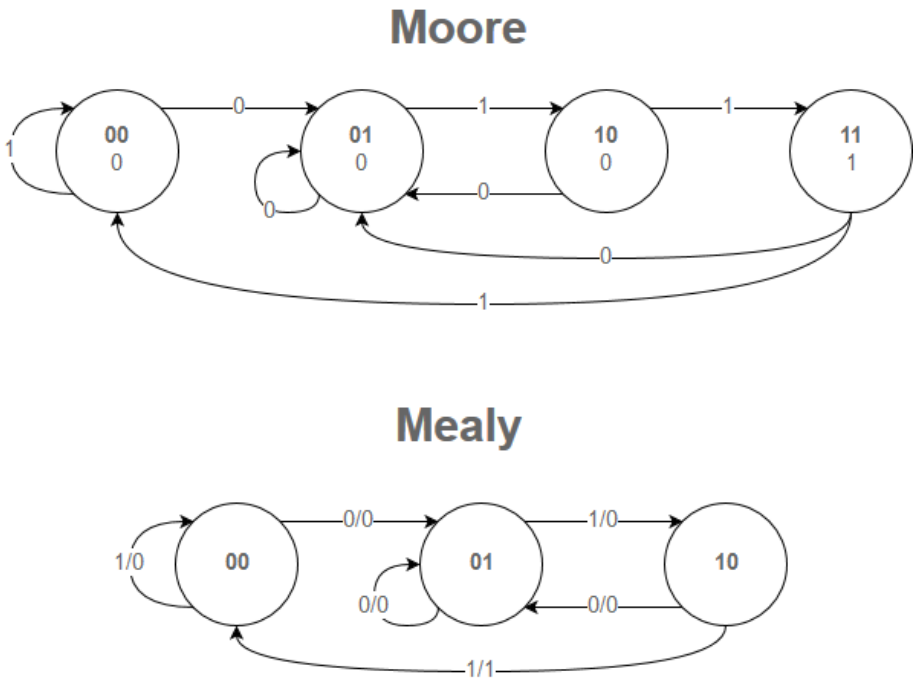
		Name	Data type	Default value	Retain
1		Input			
2		enable	Bool	false	Non-retain
3		time2go	Time	T#0ms	Non-retain
4		Output			
5		out	Bool	false	Non-retain
6		time	Time	T#0ms	Non-retain
7		output_led	Struct		Non-retain
8		1	Bool	false	Non-retain
9		2	Bool	false	Non-retain
10		3	Bool	false	Non-retain
11		InOut			
12		reset	Bool	false	Non-retain
13		Static			
14		<Add new>			
15		Temp			
16		led	Int		

Obr. 4-Proměnné bloku Timer

2.4.10 Stavový automat v jazyku SCL

Stavový automat je teoretický model jednoduchého počítače, který může nabývat předem daný konečný počet jednotlivých stavů. Stavové automaty se vyznačují, že jediná použitá paměť je jejich aktuální stav. Známe dva typy základních stavových automatů a to Mealyho a Moorův. [12]

Na obrázku 9. shora vidíme diagram Moorova automatu. Výstupy jsou určené pouze vnitřním stavem a nezávisí na vstupním stavu. Nad šipkou vidíme, do jakého stavu vedou jednotlivé proměnné. Reakce na vstup je u Moorova automatu vidět až v následujícím kroku.



Obr. 5-Stavový automat Moore/Mealy[9]

Na obrázku 9. dole vidíme diagram Mealyho automatu.. Nad šipkou vidíme do jakého stavu vedou jednotlivé kombinace proměnných. Mealyho reakce na vstup je okamžitá.

2.4.10.1 Modelový příklad

V modelovém příkladu vidíme stavový automat, který detekuje přítomnost dvou jedniček po sobě na prvním vstupu, které jsou synchronizovány dle hodinového signálu z druhého vstupu. Pokud kdykoliv v průběhu procesu přijde na první vstup nula, stavový automat se vrací do nultého stavu. Pokud již přišly dvě jedničky po sobě a přijde další, tak stavový automat zůstává ve stavu dva. Posledním bodem je ošetření nespecifikovaného stavu-v tomto případě se automat přepne do stavu nula.

```
CASE #state OF
  0:  "R_TRIG_DB_1"(CLK := #clk,
                    Q => #clock);
    IF (#in=1)AND(#clock)  THEN
    #state := 1;
    #clock := 0;
      ;
    END_IF;
```

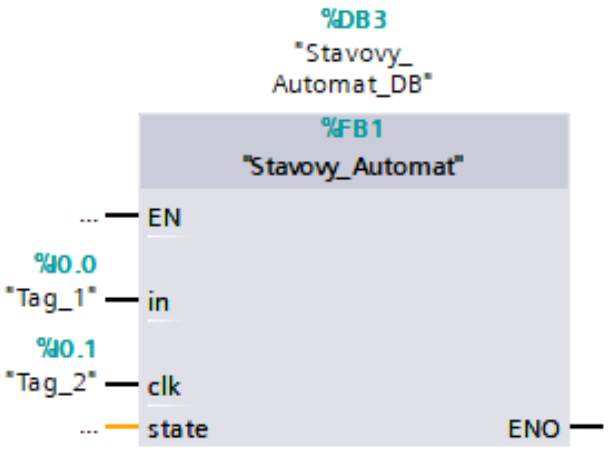
```
    ;
  1:  "R_TRIG_DB_1"(CLK := #clk,
                    Q => #clock);
      IF (#in = 1) AND (#clock) THEN
        #state := 2;
        #clock := 0;
        ;
      ELSE #state:=0;
      END_IF;
      ;
  2: "R_TRIG_DB_1"(CLK := #clk,
                    Q => #clock);
      IF (#in = 1) AND (#clock) THEN
        #state := 2;
        #clock := 0;
        ;
      ELSE
        #state := 0;
      END_IF;
      ;
  ELSE
      #state := 0; // Statement section error=reset to state 0
      ;
END_CASE;
```

Stavovy_Automat

		Name	Data type	Default value	Retain	Accessible f...	Writa...
1		Input					
2		in	Bool	false	Non-retain	☑	☑
3		clk	Bool	false	Non-retain	☑	☑
4		Output					
5		output	Bool	false	Non-retain	☑	☑
6		InOut					
7		state	Int	0	Non-retain	☑	☑

Obr. 6-Proměnné bloku stavový automat

Na obrázku 6. se nachází seznam proměnných funkce Stavovy_automat v SCL, reprezentující blok stavového automatu. Celý stavový automat je zapouzdřený v konstrukci CASE, která zajišťuje přepínání stavů podle proměnné *state* reprezentující aktuální stav. Na začátku každého stavu sledujeme náběžnou hranu signálu clk a pokud ji detekujeme a zároveň je na prvním vstupu jednička, tak se posouváme do dalšího stavu.



Obr. 7-Hotový blok stavový automat

Na obrázku 7. výše je použití bloku stavového automatu v hlavním bloku OB1. Na oranžově znázorněném pinu state je vyveden aktuální stav stavového automatu, který vychází z přivedených signálů I0.0(in) a I0.1(clk).

2.4.11 Komunikace MODBUS v jazyku SCL

V PLC Siemens můžeme dobře využít standart průmyslové komunikace MODBUS přes rozhraní PROFINET. V TIA Portal 14 na to jsou určené předpřipravené funkce MB_SERVER a MB_CLIENT.

2.4.11.1 Klient

Na kódu níže se nachází funkce MODBUS klienta napsaná v SCL. Pro její obsluhu je výhodné použít připravenou datovou strukturu MB_CLIENT. Skládá se ze základních proměnných potřebných pro jeho běh a to jsou:

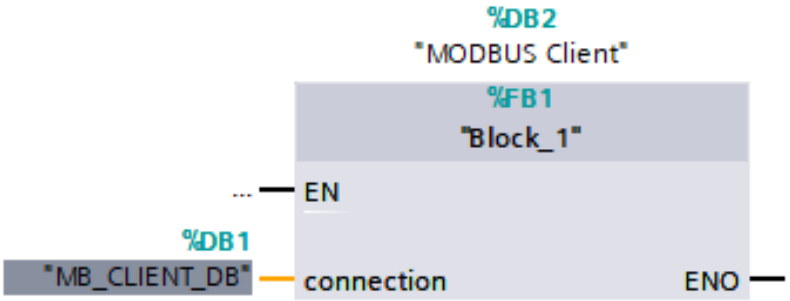
- REQ-Je vyžadováno spojení
- DISCONNECT-Pokud je v logické 1-spojení se přeruší
- MB_DATA_ADDR-Kombinace této proměnné a proměnných MB_DATA_LEN a MB_MODE určuje, jaký kód MODBUS funkce použije pro přenos
- MB_MODE- Mód přenosu
- MB_DATA_LEN-Délka dat
- DONE-Přenos hotov
- BUSY-Přenos probíhá
- ERROR-Kód chyby, pokud se vyskytla
- STATUS-Aktuální status MODBUS klienta

- MB_DATA_PTR-Pointer ukazující na data, která se mají přijmout a nebo odeslat
- CONNECT-Údaje určující ke kterému hostiteli se má připojit

```
"MB_CLIENT_DB"(REQ:=#connection.REQ,
               DISCONNECT:=#connection.DISCONNECT,
               MB_MODE:=#connection.MB_MODE,
               MB_DATA_ADDR:=#connection.MB_DATA_ADDR,
               MB_DATA_LEN:=#connection.MB_DATA_LEN,
               DONE=>#connection.DONE,
               BUSY=>#connection.BUSY,
               ERROR=>#connection.ERROR,
               STATUS=>#connection.STATUS,
               MB_DATA_PTR:=P#M1000.0 WORD 500,
               CONNECT:=#connection.TCON.ID);
```

1	Input			
2	<Add new>			
3	Output			
4	<Add new>			
5	InOut			
6	connection	MB_CLIENT		
7	Input			
8	REQ	Bool	Activates the requested transmission if TF	
9	DISCONNECT	Bool	Initiates a disconnect operation	
10	MB_MODE	USInt	Specifies the type of request: read, write (	
11	MB_DATA_ADDR	UDInt	Specifies the starting address of the data	
12	MB_DATA_LEN	UInt	Specifies the number of bits or words to b	
13	Output			
14	DONE	Bool	Instruction finished without error	
15	BUSY	Bool	Modbus transaction in progress	
16	ERROR	Bool	Instruction finished with error	
17	STATUS	Word	Detailed error information	
18	InOut			
19	MB_DATA_PTR	Variant	Reference to the local source or destinati	
20	CONNECT	Variant	Reference to the connection parameters	
21	Static			
22	Static			

Obr. 8-Proměnné bloku MODBUS client



Obr. 9-Hotový blok MODBUS client

2.4.11.2 Server

Na obrázku níže se nachází funkce MODBUS serveru napsaná v SCL. Pro její obsluhu je výhodné použít připravenou datovou strukturu MB_SERVER. Skládá se ze základních proměnných potřebných pro jeho běh a to jsou:

- DISCONNECT-v případě přivedení logické jedničky server odpojí veškerou probíhající komunikaci a další již nenaváže.
- NDR-Příznak nově zapsaných dat
- DR-Příznak nově čtených dat
- ERROR-Chyba v nastavení, či komunikaci
- STATUS-Aktuální status MODBUS serveru
- MB_HOLD_REG-Datový registr, do kterého má daný server přístup. Můžeme nastavit jednotlivé části paměti a nebo celou paměť.

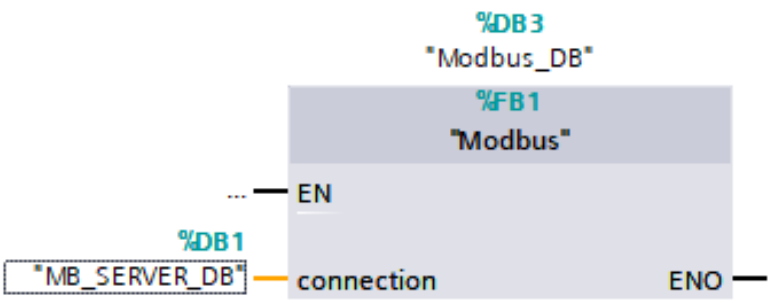
- CONNECT-Detaily nastavení komunikace-Můžeme například přijímat komunikaci pouze z určité IP adresy a tím zvýšit bezpečnost.

Modbus

		Name	Data type	Default value	Retain
5		InOut			
6		connection	MB_SERVER		
7		Input			
8		DISCONNECT	Bool		Non-retain
9		Output			
10		NDR	Bool		Non-retain
11		DR	Bool		Non-retain
12		ERROR	Bool		Non-retain
13		STATUS	Word		Non-retain
14		InOut			
15		MB_HOLD_REG	Variant		
16		CONNECT	Variant		
17		Static			
18		TCON	TCON		
19		TSEND	TSEND		

Obr. 10-Proměnné bloku MODBUS server

```
"MB_SERVER_DB"(DISCONNECT:=#connection.DISCONNECT,
               NDR=>#connection.NDR,
               DR=>#connection.DR,
               ERROR=>#connection.ERROR,
               STATUS=>#connection.STATUS,
               MB_HOLD_REG:="DB1".MODBUS,
               CONNECT:=#connection.TCON.ID);
```



Obr. 11-Hotový blok MODBUS server

3 PLC SIMATIC S7-1500

Programovatelný automat Siemens Simatic S7-1500 byl řídícím systémem v předchozí úloze a zůstane zachován i v úloze inovované. Jedná se o výkonný, modulární automat, plně kompatibilní s TIA PORTAL 14. Důležitou vlastností oproti starším automatům je podpora jazyku GRAPH(Grafcet) a SCL ve kterém byla naprogramováno řízení jednotlivých fází předcházející úlohy. [2]

3.1 Rozšiřující karty

Programovatelný automat jako celek je tvořený hlavní procesorovou jednotkou, zdrojem a jednotlivými přídavnými kartami. Díky této modularitě si můžeme sestavit PLC podle nároků jednotlivých aplikací.

3.2 PLC použité v úloze

CPU 1512C-1 PN-Jednotka má dva Profinet konektory, kapacitu 250 kB pro program a 1 MB pro data, což pro naše účely bohatě postačuje.

Obr. 12-Možné konfigurace PLC S7-1500 [4]

3.3 Zdroj

Typ PS-25W-12VDC- Stabilizovaný zdroj napájející soustavu samotného PLC a přídavných karet přes zadní sběrnici.

3.4 AI 5/AQ 2

Karta analogových vstupů a výstupů. Obsahuje 5 kanálů analogových vstupů a 2 kanály analogových výstupů.

3.5 DI 16/DQ 16_1

Karta digitálních vstupů a výstupů. Obsahuje 16 kanálů digitálních vstupů a 16 kanálů digitálních výstupů.

3.6 DI 16/DQ 16_2

Druhá karta obsahuje stejně, jako předešlá karta digitálních vstupů a výstupů 16 kanálů digitálních vstupů a 16 kanálů digitálních výstupů.

4 ZÁVĚR

Hlavním cílem této bakalářské práce bylo vypracovat dokumentaci programu v PLC k předmětu MAUP podle aktuálního TIA Portálu 14 SP1. [Příloha 1] Dokumentace úlohy je připojena jako příloha. Připojen je také stručný výpis kódu pro potřeby opravování vypracovaných projektů vyučujícím. [Příloha 2] Jakýmsi bonusem pro studenty, kteří budou tuto úlohu vypracovávat je „tahák“ se základní syntaxí jazyka SCL, který bude na pracovišti jako součást úlohy. [Příloha 3] Zadání se mi podařilo i přes problémy s některými částmi programu splnit. Aktuální vývojové prostředí se velmi liší od toho, ve kterém je navržena prapůvodní úloha, kterou jsem přepracovával v rámci semestrální práce a proto musely být některé instrukce upraveny, či nahrazeny obdobnými. Při tvorbě semestrální práce předcházející této bakalářské práci jsem si osvěžil učivo předmětu BPGA, které se mi při zpracovávání velmi hodilo. Jednotlivé bloky byly v semestrální práci psány jazyky LAD, STL a GRAFCET podle toho, který byl pro daný účel nejvhodnější.

Na rozdíl od toho je možné pro PLC v této bakalářské práci psát celý program pouze v programovacím jazyku SCL, který mi po plném osvojení vyhovuje mnohem více, než předcházející jazyky, které byly v různých ohledech více limitovány. Oproti tomu jazyk SCL disponuje vším pro komfortní napsání celého programu a odpadá tímto rozhodování, který programovací jazyk pro ten konkrétní blok zvolit.

K programu v PLC byla vytvořena i vizualizace v HMI, která s PLC komunikuje přes Ethernet. HMI se skládá ze šesti různých obrazovek-výchozí obrazovky, která podává základní informace o procesu. Můžeme odtud řídit jednotlivé aktory jako například ventily, míchadlo a zahřívání. Všechny můžeme zapnout/vypnout a přepínat do manuálního/automatického módu. V dalších obrazovkách již ovládáme jednotlivé fáze, popřípadě parametry procesu.

Literatura

[1]*PLC Simatic S7-1500 Manual* [online] [cit. 2017-12-25]. Dostupné z: https://www.automation.siemens.com/salesmaterial-as/interactive-manuals/getting-started_simatic-s7-1500/documents/EN/software_complete_en.pdf

[2] *PLC Simatic S71200, S7-1500* Programming guideline[online] [cit. 2017-12-25]. Dostupné z: http://www1.siemens.cz/ad/current/content/data_files/automatizacni_systemy/mikrosystemy/simatic_s71200/programming-guideline-for-s71200-s71500_2014-09_en.pdf

[3] S7-GRAPH V5.3 for S7-300/400 Programming Sequential Control Systems [online] [cit. 2017-12-26]. Dostupné z: https://cache.industry.siemens.com/dl/files/630/1137630/att_28560/v1/Graph7_e.pdf

[4] Tisková zpráva S7-1500[online] [cit. 2017-12-26]. Dostupné z: http://www.siemens.cz/press/nove-kompaktni-ridici-jednotky-rady-simatic-s7-1500

[5] *Structured Text Tutorial to Expand Your PLC Programming Skills* [online]. [cit. 2018-05-20]. Dostupné z: http://www.plcacademy.com/structured-text-tutorial/#repeat-loops

[6] *Structured Text Tutorial to Expand Your PLC Programming Skills* [online]. [cit. 2018-05-10]. Dostupné z: http://www.plcacademy.com/structured-text-tutorial/#for-loops

[7] *Structured Text Tutorial to Expand Your PLC Programming Skills* [online]. [cit. 2018-05-10]. Dostupné z: http://www.plcacademy.com/structured-text-tutorial/#while-loops

[8] K. HO, Anthony. *Structured Text Programming* [online]. **2012** [cit. 2018-05-10]. Dostupné z: https://pdhonline.com/courses/e334/e334content.pdf

[9] *State machines* [online]. [cit. 2018-05-10]. Dostupné z: https://www.norwegiancreations.com/wp-content/uploads/2017/03/moore_mealy.png

[10] *Structured Text (ST) Programming Guide Book* [online]. [cit. 2018-05-10]. Dostupné z: http://dl.mitsubishielectric.com/dl/fa/document/manual/plc/sh080368e/sh080368eh.pdf

[11] PÁSEK, Jan a Václav KACZMARCZYK. *Automatizace procesů Laboratorní cvičení I*. BRNO: VUT FEKT [cit. 2018-05-10]

[12] *What is a state machine?* [online]. [cit. 2018-05-10]. Dostupné z: https://www.techopedia.com/definition/16447/state-machine

Seznam příloh

Příloha 1. ŠKAPA, A. *Inovace laboratorní úlohy - "Tanky"-nová dokumentace.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2018. Vedoucí bakalářské práce Ing. Václav Kaczmarczyk, Ph.D..

Příloha 2. ŠKAPA, A. *Inovace laboratorní úlohy "Tanky"-výpis kódů.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2018. Vedoucí bakalářské práce Ing. Václav Kaczmarczyk, Ph.D..

Příloha 3. ŠKAPA, A. *Inovace laboratorní úlohy "Tanky"-základní syntaxe SCL.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2018. Vedoucí bakalářské práce Ing. Václav Kaczmarczyk, Ph.D..